View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Servicio de Coordinación de Bibliotecas de la Universidad Politécnica de Madrid

A Novel SDN based Stealthy TCP Connection Handover Mechanism for Hybrid Honeypot Systems

Wenjun Fan and David Fernández
Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
Madrid, Spain 28040
Email: {efan,david}@dit.upm.es

Abstract—**Honeypots have been largely used to capture and investigate malicious behavior through deliberately sacrificing their own resources in order to be attacked. Hybrid honeypot architectures consisting of frontends and backends are widely used in the research area, specially due to the benefits of their high scalability and fidelity for detailed attacking data collection. A hybrid honeypot system often needs a facility aimed to tightly control the network traffic, for purposes such as redirecting the traffic from the frontends to the backends for in-depth attack analysis. However, the current traffic redirection approaches, particularly the TCP connection handover mechanisms, are not stealthy and they can be easily detected by attackers. This paper proposes an SDN based network data controller for hybrid honeypot systems that uses a transparent TCP connection handover mechanism and provides a traffic filtering approach based on the Snort alert functionality. The controller is implemented as an application based on the open-source Ryu SDN framework. It allows the users to configure their own network data control rules, which based on the Snort alert messages will forward or redirect the traffic to the corresponding honeypots. The experiments validate the proposed mechanism and the testing results show that the controller can efficiently perform the stealthy TCP connection handover as well.**

Index Terms—**Honeypots, Cyber Security, Virtualization, SDN, Traffic Redirection, Intrusion Detection**

I. Introduction

A honeypot is a security tool created to be attacked with the aim to capture the attack datasets for measuring and investigating network threats. Though an important number of independent honeypot software have been developed [1], according to the interaction level, the honeypots can be roughly classified into three categories: low-interaction, medium-interaction and high-interaction honeypots. Low-interaction honeypots (LIH) like Honeyd [2] can emulate multiple decoys simultaneously to monitor unauthorized traffic. These decoys can emulate the appearance of operating systems and vulnerable services, but they provide little interaction to the adversaries. Medium-interaction honeypots (MIH) like Dionaea [3] can provide much more interaction capacity to the adversaries, allowing to catch the malicious payload. They can emulate a variety of vulnerable services by using the TCP/IP network protocols implemented and managed by the underlying operating system where the MIH runs. However, they are limited by the fact that they just emulate well known vulnerabilities, being its security program focused on capturing the malicious traffic accessing to that emulated vulnerable services. Finally, a genuine computer system running as a honeypot is called high-interaction honeypot (HIH), since it can provide a fully functional operating system to the adversaries. Using HIHs, security researchers can capture not only the network activity, but also the system activity. However, the limitation of HIHs is the high resource consumption when used for large-scale deployment.

Therefore, depending on its interaction level, these independent honeypots are either expensive to administrate and poorly scalable (HIHs) or based on emulated resources that limit the level of detail they can collect about attacks (MIHs and LIHs). Besides, the usage of single independent honeypot is not adequate to cope with the various attacking scenarios. In order to address the drawbacks, the hybrid honeypot architecture and a number of relevant hybrid honeypots [4], [5], [6] were proposed. A hybrid honeypot system often include multiple frontends (i.e. LIHs or MIHs) and backends (i.e. HIHs). The frontends are often deployed in large-scale network space to provide highly scalable data collection, while the backends are often deployed in a centralized honeyfarm for in-depth data analysis.

Consequently, hybrid honeypots combine in one complex system the benefits of both types of honeypots: the high scalability of LIH/MIH and the fidelity of HIH. However, hybrid honeypot systems require the use of a mechanism to redirect the traffic from the frontends to the backends. The traffic redirection functionality provided by current tools is often not transparent enough to avoid the adversary to easily detect it. This fact will go against the data capture utility of the honeypot, because once the adversary is aware of his submergence in a honeypot environment, he will stop the attack and even pollute or damage his tracks to avoid being traced.

Hence, new mechanisms to transparently redirect the traffic from the frontends to the backends are needed to be implemented in network data controllers of hybrid honeypot systems. In this paper, a novel SDN controller application is proposed for being used in hybrid honeypot systems, that provides traffic filtering capabilities for data reduction and a traffic redirection mechanism for transferring interesting traffic

978-1-5090-6008-5/17/$31.00 © 2017 IEEE

connections according to different requirements of intrusion investigations.

The remainder sections of this paper are organized as follows: Section 2 reviews the related work about hybrid honeypot systems and traffic redirection approaches; Section 3 proposes the design of the controller's architecture and its functions; Section 4 describes the implementation of the prototype; Section 5 presents the experiments and shows the testing results; finally, Section 6 presents some conclusions and suggest future work.

II. Related Work

Since 2006 a number of hybrid honeypot systems have been staged [4], [5], [6] to take advantage of the benefits of the hybrid approach: the collection of datasets of detailed attack processes covering large network address spaces. For example, the hybrid honeypot framework described in [7] integrates the well-known virtual LIH Honeyd [2] with the Gen III honeynet architecture [8] for improving intrusion detection systems that protect local production networks.

As stated in the introduction, one of the main parts of a hybrid honeypot system is the traffic redirection mechanism, which is aimed to connect the frontends and the backends, filtering and redirecting the interesting traffic to the high-interaction honeypots for further investigation. Several approaches have been proposed to facilitate this function. The hybrid honeypot framework [7] simply used the Honeyd's built-in proxy function to redirect the traffic into high-interaction honeypots. However, because of the lack of a traffic filtering mechanism in this approach, the backends could be easily flooded with invalid data. Besides, this simple proxy approach suffers from the identical-fingerprint problem, since the frontends and the backends have different IP addresses assigned. Some other hybrid honeypot systems [4], [9] use GRE tunnel to connect the frontends and the backends. Due to the fact that backends appear to be directly deployed in the production network, this type of hybrid honeypot systems do not suffer from the identical-fingerprint problem. But all the traffic is only either discarded or forwarded, owing to the fact that frontends have no interaction capability, preventing the implementation of more intelligent filtering capabilities.

Bailey et al. [5] use a connection handover mechanism for traffic redirection. In order to avoid saving the state of every TCP connection, the connection handoff mechanism makes the decision based on the first payload of any conversation. However, the author does not reveal the technical details about how the connection handoff is made. Similarly, Honeybrid gateway [10] uses a connection replay approach to implement traffic transparent redirection between LIH and HIH. Nevertheless, Honeybrid reveals the technical detail of the gateway, which uses a TCP replay proxy based approach that makes the use of Linux libnetfilter_queue [11] to process packets. In this case, knowing the details of the mechanism used, security researchers can gain more insight into how connection handoff is made.

Recently, some researchers have proposed solutions to address the identical-fingerprint problem in hybrid honeypot systems. For example, Lengyel et al. proposed a hybrid honeynets architecture namely VMI-Honeymon [12], which provides a novel solution to the clone-routing problem. The challenge is to manage network connectivity with identical HIHs clones without any internal in-guest network reconfiguration, because the network interfaces in each clone will have to remain identical, sharing the same MAC and IP address of the original VM. Having the same MAC and IP addresses will cause collision if the clones are placed on the same network bridge. But in-guest network reconfiguration would inadvertently lead to changing the state of the HIH. Thus, the solution proposed retains the MAC and IP address of the original VM for each clone, and each clone is placed upon a separate network bridge which is attacked to the VM running Honeybrid gateway, providing isolation for the MAC and avoiding the collision. This solution indirectly addresses the identical-fingerprint problem.

In addition, Fan et al. [13], [14] proposed a hybrid honeypots based traffic redirection mechanism intending for addressing the identical-fingerprint problem. The idea behind the proposal is promising, while the solution still has some drawbacks. Different honeypots using the identical-fingerprint have to frequently switch up and down according to research requirements. Besides, it is hard to conduct large-scale deployment using the proposed hybrid architecture.

Software defined networking (SDN) is a new networking paradigm that is aimed to separate the functions that determine the direction of traffic (control plane) from the underlying systems that forward traffic to the selected destination (data plane). SDN brings a fine flow control capability and programmability to networks that allows to dynamically configure the data plane according to the network administrator requirements. These SDN advantages perfectly match the requirements of traffic redirection mechanisms. For example, Binder et al. [15] proposed an SDN based method for TCP connection handover in a Content Delivery Network to deliver the requests into the best server.

At present, the SDN technology has been widely used in the research field of network security in distributed systems [16], [17]. The NICE system [18] is indeed an SDN based intrusion response system (IRS), which includes a number of attack countermeasures. It sufficiently makes use of the network data and applies the scenario attack graph (SAG) as well as the network intrusion detection system (NIDS, i.e. Snort [19]) to analyze the network activity in order to detect the suspicious and even the compromised VMs. Because it focuses on detecting the zombies or botnets for preventing DDoS attacks, the attack countermeasures mainly consisting of network reconfiguration, packet manipulation, and simple traffic containment, which can be easily achieved using SDN technology. The fact that the NICE does not use honeypots but the production VMs makes it is hard to detect the anomaly-based attacks, i.e. zero-day attacks. Thus, it is unable to reduce the negative false alarm. Therefore, the

NICE is powerful to confine the spread of the well-known signature-based attacks among production systems, while it has limitations to investigate anomaly-based attacks.

III. The Design of the Controller Application

Figure 1 shows an overview of the proposed software architecture for the network data controller. It is mainly organized around an OpenFlow based switch which manages the control plane and it is in charge of redirecting the connections among the controller and the different honeypots, and a Ryu SDN framework based controller application, which includes a decision and a redirection engine and makes use of an IDS.

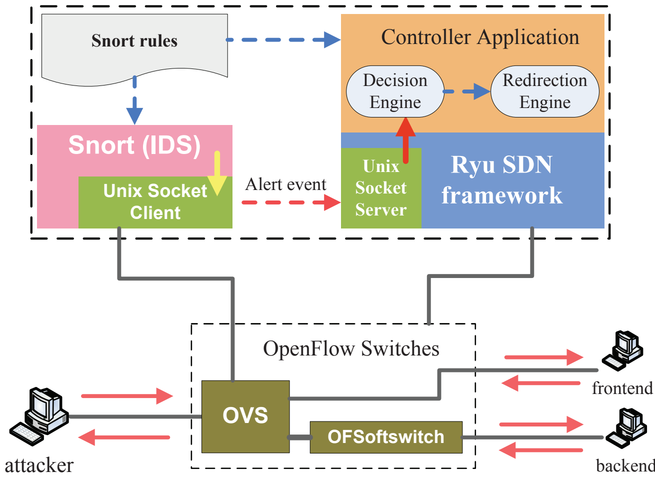

Fig. 1. The Architecture of the Controller Application

On one hand, in the control plane, the open-source IDS Snort is used to analyze the traffic to generate alerts and send the alert messages to the controller application through UNIX socket. According to the alert message, the decision engine (DE) will take the decision to forward or redirect connection and signal the redirection engine (RE) to perform the corresponding action.

On the other hand, in the data plane, there are two types of OpenFlow Switches. The main switch, named the Flow Classification Forwarder (FCF), is an Open vSwitch (OVS) that manages the traffic among the different components of the system (controller, honeypots and Snort IDS). The frontend and the backend honeypots use the same IP and MAC addresses, but they are connected to different out ports of the FCF, which are used to distinguish them. This aims to ensure the transparent TCP connection handover will not be detected by fingerprinting. The second switch, a modified OFSoftswitch named Session Process Forwarder (SPF), is used to implement the sequence (Seq) and acknowledgement (Ack) number synchronization function needed for the TCP redirection mechanism to work. A SPF is deployed in front of any backend in order to synchronize the Seq and Ack number of the TCP connections being redirected to it.

The SDN controller based TCP connection handover mechanism and the Snort based traffic filtering approach are described in detail in the next two subsections.

A. TCP Connection Handover Mechanism

This subsection focuses on presenting the redirection engine's function, which mainly provides the SDN based TCP connection handover mechanism. Figure 2 graphically describes the three phases that make up the mechanism, whose details are described as follows.

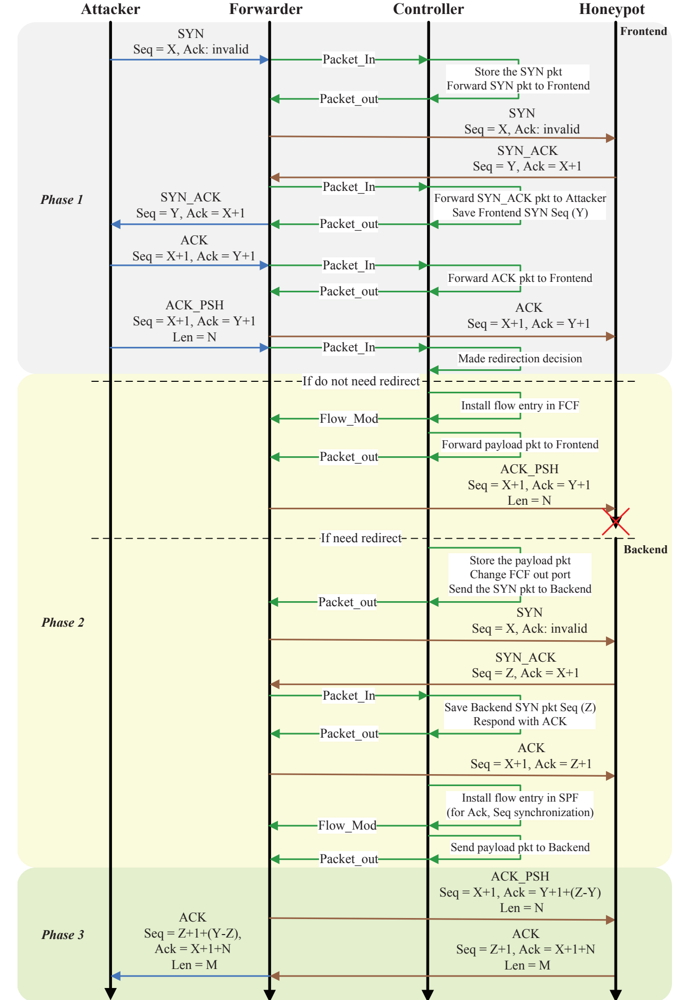

Fig. 2. Transparent TCP connection handover mechanism

Phase 1: The session between the attacker and the controller is established.

The attacker sends the TCP connection request to the target honeypot. The controller will firstly forward the request to the frontend, allowing the frontend and the attacker to perform the TCP three way handshake to establish the connection. In this phase, we assume that the initial Seq number chosen by the attacker is X, and the one chosen by the frontend is Y. Therefore, the Seq and Ack numbers of both the attacker and the frontend once the connection has been established are:

$$Attacker : Seq = X + 1, Ack = Y + 1 \tag{1}$$

$$Frontend : Seq = Y + 1, Ack = X + 1 \tag{2}$$

Apart from establishing the connection, during this phase the application has to: store the SYN packet received from the attacker just in case it needs to replay it later in order to transfer the connection; save the initial Seq number (X) of the SYN packet from the attacker; and save the initial Seq number (Y) of the SYN_ACK packet from the frontend. Thereafter, when the payload packet from the attacker is received, the controller has to save and inspect it, so as to make a decision about whether the connection is worth being redirected to the backend, if it is still kept in the frontend, or if it should be closed. As mentioned before, Snort alerts are used to make the decision. The details of the decision engine main function will be presented in the next subsection.

Phase 2: TCP session is being transferred from the frontend to the backend using a TCP replaying approach.

If the decision is to redirect the connection, the controller's redirection engine starts to replay the three-way handshake by sending the attackers SYN packet saved in the first phase to the backend. Then, the backend honeypot will respond with a SYN_ACK packet including a randomly chosen Seq number, which we assume to be Z. After the controller receives the SYN_ACK packet from the backend, it will perform two actions: save the initial Seq number Z of the backend; and answer the SYN_ACK packet with the ACK packet that is saved in the first phase as well. After that, the replaying of the three-way handshake between the controller and the backend is finished and the TCP connection is now formally established between them.

At this moment, the Seq and Ack numbers of both the controller and the backend are:

$$Controller : Seq = X + 1, Ack = Z + 1 \tag{3}$$

$$Backend : Seq = Z + 1, Ack = X + 1 \tag{4}$$

Later, the saved payload packet has to be sent from the controller to the backend. However, before sending it, some actions have to be done for synchronizing the Seq and Ack numbers.

Notice that the backend expects to receive the payload packet that has the same Seq and Ack numbers calculated in equation (3). But the preserved payload packet has the Seq and Ack numbers that are calculated in equation (1). Due to the difference in the Ack number between equation (1) and (3), in order to synchronize the Ack number, we will have to modify the Ack number in the payload packet by a D-Ack value that can be calculated by equation (5):

$$D\text{-}Ack = Z - Y \tag{5}$$

Therefore, the first payload packet's Ack number (Y+1) have to be increased by the D-Ack value (Z-Y)to be transformed to the one (Z+1) that the backend honeypot excepts.

After sending the modified payload packet, the honeypot will acknowledge it. If we suppose that the data length of the payload packet is N, then the ACK packet sent by the honeypot should have the Seq and Ack number calculated as expressed by equation (6):

$$ACKfromBackend : Seq = Z + 1, Ack = X + 1 + N \tag{6}$$

However, the attacker expects to receive the ACK packet from the honeypot with the Seq and Ack numbers as follows:

$$ACKfromController : Seq = Y+1, Ack = X+1+N \tag{7}$$

The difference between the two above equations is the value that has to be added to the Seq number of the segments going to the attacker. We denote the difference Seq value as D-Seq, which can be calculated by equation (8):

$$D - Seq = Y - Z \tag{8}$$

Once the D-Ack and D-Seq have been calculated, the values have to be communicated to the SPF, which is the element in charge of performing the sequence numbers synchronization. A modified OFSoftswitch that includes a new specific function to modify the TCP Seq and Ack numbers, as well as some new OpenFlow options to manage it in FlowMod messages has been used for this purpose. Following this approach, the controller will install flow entries in the SPF to perform the synchronization: each packet coming from the backend will have the D-Seq added to its Seq number, and each packet going to the backend will have the D-Ack added to its Ack number. Once it is done, the saved payload segment is replayed to the backend.

Phase 3: TCP session has been transferred to the backend and Seq and Ack numbers are being synchronized.

After the changes in phase 2, the connection has been formally transferred and the rest packets have to be exchanged directly between the attacker and the backend. For this purpose, the corresponding flow entries will be installed into the OVS. Therefore, all the subsequent segments will avoid being forwarded to the controller and can smoothly traverse between the two TCP end points.

As a summary, the algorithm of the SDN based TCP connection handover used by the controller and described in this subsection is presented as Algorithm 1.

B. Traffic Filtering Approach

This subsection presents the traffic filtering functionality of the system provided by the cooperation of the decision engine and the Snort alert function. We have reused the Snort rule format as the basis to set our own traffic-control rules. Roughly, Snort alert rule format is as follows:

> **alert** *protocol source-ip source-port* $\rightarrow$ *destination-ip destination-port* (**msg**: "alert message"; **sid**: *an integer*; **priority**: *an integer*; **content**: "malicious pattern";)

The text using bold font is the key words, and the text using italics font needs to be replaced by the values. Thus, an example of Snort NIDS rule can be:

Algorithm 1 SDN controller's redirection engine

```
Require: SDN Controller and switches
    SDN Controller Initialisation :
 1: SDN switches are configured
    PacketIn Event Loop Process :
    {SDN controller waits for PacketIn event}
 2: while PacketIn event do
 3:   if PacletIn is inbound then
 4:     if pkt is SYN pkt then
 5:       if pkt not in sessions{} then
 6:         new sessions[pkt.ipv4_src, pkt.tcp_src]
 7:       end if
 8:       save SYN pkt
 9:       respond with SYN_ACK pkt
10:       save Seq number Y
11:     end if
12:     if pkt is ACK pkt then
13:       if pkt has no payload data then
14:         wait for the next pkt
15:       end if
16:       if pkt has payload data then
17:         store payload pkt
18:         make decision: DROP or REDIRECTION
19:         if DROP then
20:           discard the pkt
21:           end the connection
22:         end if
23:         if REDIRECTION then
24:           set FCF's out port linking backend
              {begin TCP connection replaying}
25:           send the saved SYN pkt to the out port
26:         end if
27:       end if
28:     end if
29:   else if PacketIn is outbound then
30:     if pkt is SYN_ACK pkt then
31:       save Seq number Z
32:       respond with ACK pkt
33:       calculate D-Ack = Z - Y
34:       calculate D-Seq = Y - Z
35:       install flow entries in SPF
36:       send saved payload pkt to honeypot
37:     end if
38:   end if
39: end while
```

```
alert tcp any any → 192.168.1.0/24 111 (msg:"external mountd access")
```

This rule means that any TCP traffic that wants to access the port 111 of a system in the network 192.168.1.0/24 will match this signature, and the Snort will make an alert with the message "external mountd access". Furthermore, we can make the rule more specific by adding the checking of a suspicious pattern to the rule, e.g.:

```
alert tcp any any → 192.168.1.0/24 111 (msg:"external mountd access"; content:"|000186a5|")
```

Now the signature includes not only the IP header information but also a malicious payload pattern. Therefore, Snort will check both the IP header information and the payload data of the captured packet against this signature. When more than one rule exists, we can use the priority field to set the priority level for every rule and the sid field to set the alert ordering.

Therefore, based on the "alert" rule format of Snort, we create our own rules by setting an action into the "msg" field. In our case, we define three actions: DROP, MIH and HIH. "DROP" refers to discard the packet, while "MIH" and "HIH" indicates that the traffic destination should be the frontend and the backend respectively.

To implement the traffic control mechanism, we apply the Snort rules by two steps to carry out the traffic filtering approach. In the first step, Ryu controller will read and parse the Snort rules, translate them into flow entries, and install those flow entries into the main OVS switch during the system initialization phase. This step aims to set a data reduction measure to improve the data capture efficiency. Only the rules with a "DROP" action that do not check the "content" field are translated into "drop" flow entries. Therefore, any traffic that matches these flow entries will be efficiently discarded in the data plane, avoiding them to reach to the controller. The rest rules will be translated into "allow" flow entries, and the traffic that matches them will be forwarded as a PacketIn event to the Ryu controller for processing. In the second step, the Ryu controller will cooperate with Snort to carry out the content based traffic filtering and redirection (see Fingure 3).

The controller application will send the first payload packet of each session to the Snort port. Snort thereafter will inspect the payload packet, raise an alert, and send the corresponding action as the alert event back to the controller application through Unix socket. The algorithm of the controller's decision engine can be described as Algorithm 2.

Note that some malicious activity can not be detected by only inspecting the first payload packet, but examining a higher number of payload packets. For those cases, another well-known open-source IDS Bro [20] could be used. Bro is a bit different from Snort. In a way, Bro is both a signature and anomaly-based IDS. Thus, we could use it to detect the unknown malicious traffic patterns instead of inspecting the first payload packet. However, the collaboration of Ryu controller and Bro will need more effort on TCP connection

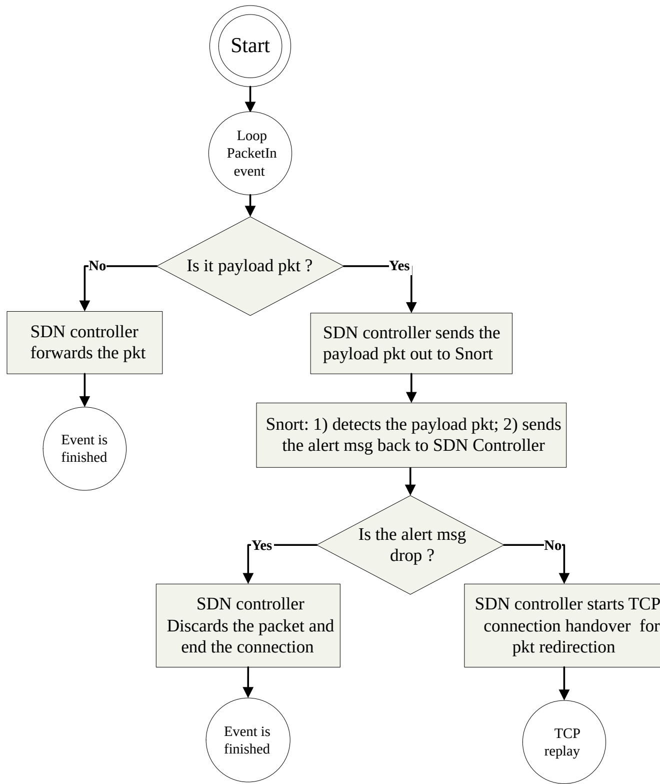

Fig. 3. The controllers workflow for making decision

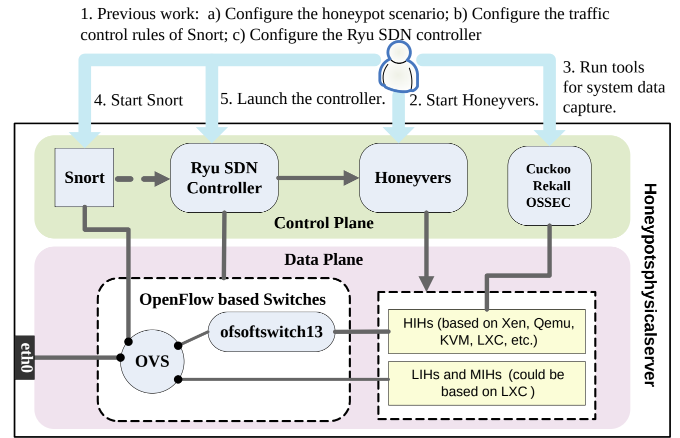

Fig. 4. An implementation for validating the controller

Algorithm 2 SDN controller's decision engine

Input: First Payload pkt
Output: Decision
Decision Making :
1: **if** pkt has payload **then**
2: Controller sends the payload pkt to Snort
3: Snort inspects the pkt
4: Snort makes the alert msg
5: Snort sends the alert msg back to the controller
6: **end if**

hand over approach, which needs to save more than one payload packets until the decision can be made and later replaying all the saved packets. At present, our controller application does not implement this function. However, we still could set a mirror port in the main OVS to mirror all the traffic to the anomaly-based IDS as off-line inspection.

IV. Implementation

In order to validate the functionality of the proposed SDN controller application, we implemented a hybrid honeypot system as a prototype that uses the proposed application as its network data controller. We implemented the whole system inside one physical machine, as it is shown in Figure 4.

Creating and configuring a honeypot network (honeynet) manually is, in general, a complex and tedious task. In order to hide all the technical dependent complexity of the underlying tools, we have used the honeypot deployment tool called Honeyvers [21] to describe and deploy the honeypot scenario. Honeyvers is a general virtual honeynet management framework that offers the security researchers the versatility to deploy heterogeneous honeypots through integrating the corresponding virtualized deployment tools for managing them. In this case, Honeyvers uses Virtual Networks over linuX (VNX) tool [22] as the underlying virtualization testbed for deploying virtual scenarios.

Honeyvers workflow can be summarized as follows:

1) The user prepares the configuration of the honeypot scenario based on the Honeyvers description language [23], configures the traffic control rules of Snort in order to set the traffic filtering and redirection mechanism and configures the SDN controller.
2) The Honeyvers tool is invoked to create the honeypot scenario according to the configuration provided.
3) The security tools are turned on to monitor the HIH and capture system activity.
4) Snort is launched to listen on the corresponding interface that is linked with the Snort out port of the main OVS.
5) The Ryu controller and application is launched to control the data plane and run the whole system.

Snort is configured in NIDS mode to utilize its intrusion detection and alert raising functionality. We use the snortlib, which can be accessed from Ryu SDN framework, to integrate in this way the Snort alert function into the controller application. The controller application combined with the function of snortlib has been developed and uploaded to Github (https://github.com/fanwj2010/ryu-honeymagic).

Furthermore, due to the specific requirement of implementing the Seq and Ack numbers synchronization function in the OFSoftswitch as discussed before, we have developed new functions in the OFSoftswitch to implement that functionality. OFSoftswitch is implemented in the user space, which greatly facilitates the modification of its code. Besides, the OFSoftswitch is open source and can be freely forked from the Github project. Currently, the OFSoftswitch is capable of handling OpenFlow messages till version 1.3. The

synchronization functions are achieved through adding a new action based on the SET_FIELD defined by the OpenFlow 1.3 standard. These two actions are SET_TCP_ACK_DIFF and SET_TCP_SEQ_DIFF that are used to modify the Ack and Seq numbers respectively. For example, if we install a flow mod in the OFSoftswitch using SET_TCP_SEQ_DIFF with argument 1000, incoming TCP connections which matches the flow entry will have its Seq number incremented by 1000 on the outgoing interface. The same action can be performed for the Ack number as well. Therefore, using correctly these new actions we are able to synchronize the TCP Seq and Ack numbers for the two traffic directions of the TCP connection. The new functions added to OFSoftswitch are also available at Github (https://github.com/fanwj2010/ofsoftswitch13).

V. EXPERIMENTS

A. Testing Scenario

Figure 5 presents the simple testing scenario used to validate the proposals described in this paper.

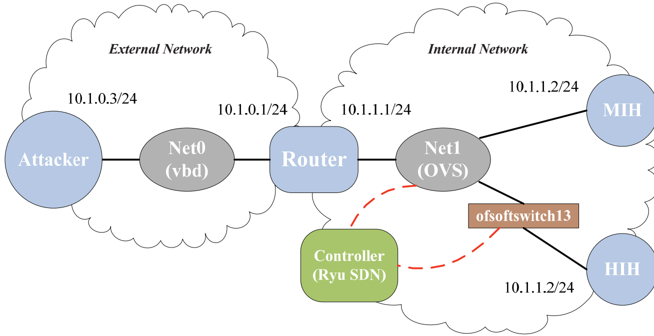

Fig. 5. The testing scenario

The testing scenario includes: one internal network (10.1.1.1/24) where the honeynet is deployed; one external network (10.1.0.1/24) where the attacker is located; and a router that connects both networks. As stated before, we assign the same IP addresses (10.1.1.2) and also the MAC address (not shown in the Figure) to both the MIH and HIH to make sure they have the identical fingerprint.

For the tests we configured the following Snort rules:

```
alert tcp any any → any 21 (msg:"MIH"; sid:1000002; priority:2;)
...
alert tcp any any → any 25 (msg:"HIH"; sid:1000005; priority:2;)
...
alert tcp any any → any any (msg:"DROP"; sid:1000008; priority:0;)
```

Therefore, the system has several open ports. The traffic destined to these ports will be sent and processed by the controller, and later the controller will forward the traffic to the destination associated to the Snort alert message. Any other uninteresting traffic will be filtered by the main OVS in order to perform data reduction, which can prevent the controller from being congestion. After the initialization of the controller application, the corresponding flow entries in the Main OVS are created as follows:

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=90.798s, table=0, n_packets=0, n_bytes=0, priority=2,tcp,tp_dst=21 actions=CONTROLLER:65535
...
cookie=0x0, duration=90.798s, table=0, n_packets=0, n_bytes=0, priority=2,tcp,tp_dst=25 actions=CONTROLLER:65535
...
cookie=0x0, duration=90.798s, table=0, n_packets=0, n_bytes=0, priority=0,tcp actions=drop
```

Once the system is started with the configuration described above, we can use traceroute from the attacker's VM to check the connectivity and find the routing path to the honeypot IP addresses.

```
# traceroute -n 10.1.1.2
traceroute to 10.1.1.2 (10.1.1.2), 30 hops max, 60 byte packets
1 10.1.0.1 0.088 ms 0.029 ms 0.025 ms
2 10.1.1.2 8.128 ms 8.021 ms 8.078 ms
```

B. Redirection

In order to validate the TCP connection handover mechanism, we use SSH to establish a TCP connection and we apply Wireshark to observe the connection changes occurred between the frontend and the backend. Figure 6 shows the observation of the TCP connection handover.

Time	10.1.0.2		10.1.1.2	Comment
4.941079	36093	SYN →	22	Seq = 0
4.941111	36093	← SYN, ACK	22	Seq = 0 Ack = 1
4.943263	36093	ACK →	22	Seq = 1 Ack = 1
4.947688	36093	← PSH, ACK - Len: 43	22	Seq = 1 Ack = 1
5.149874	36093	← PSH, ACK - Len: 43	22	Seq = 1 Ack = 1
5.353871	36093	← PSH, ACK - Len: 43	22	Seq = 1 Ack = 1
5.761863	36093	← PSH, ACK - Len: 43	22	Seq = 1 Ack = 1

Time	10.1.0.2		10.1.1.2	Comment
4.944849	36093	SYN →	22	Seq = 0
4.944876	36093	← SYN, ACK	22	Seq = 0 Ack = 1
4.951328	36093	ACK →	22	Seq = 1 Ack = 1
4.951409	36093	PSH, ACK - Len: 43 →	22	Seq = 1 Ack = 1
4.951420	36093	← ACK	22	Seq = 1 Ack = 44
4.956216	36093	ACK →	22	Seq = 44 Ack = 1
4.956278	36093	ACK →	22	Seq = 44 Ack = 1
4.957416	36093	← PSH, ACK - Len: 43	22	Seq = 1 Ack = 44
4.957858	36093	← ACK - Len: 1448	22	Seq = 44 Ack = 44
4.958339	36093	ACK →	22	Seq = 44 Ack = 44

Fig. 6. The Wireshark flow graphs of the SSH redirection testing

The upper flow graph shows the initial TCP connection established between the attacker and the frontend. However, thereafter the attacker does not sent the ACK_PSH segment to the frontend, since the Snort makes an alert indicating

the controller to redirect the traffic to the backend. Thus, the controller starts to replay the TCP three-way handshake as the lower graph shows. After finishing the new TCP establishment between the attacker and the backend, the rest of the segments are exchanged fluently between them. Meanwhile, the segments retransmitted by the frontend to the attacker are dropped by the controller, and finally the old TCP connection between the attacker and the frontend will be terminated. Here we should note that the time displayed by Wireshark is relative to the first packet it captures by the network interface it is listening on during the testing (not related to the time the connection was initiated).

C. Performance

For the performance evaluation, we designed a simple test based on SMTP to monitor the latency of the first push packets arriving at the honeypot under concurrent incoming connections. An SMTP server (Postfix) was installed in honeypots. An SMTP client script was installed on the remote attacker. The script consists of the following sequence of five SMTP commands:

```
HELO test \n
MAIL FROM: <test@test.test> \n
RCPT TO: <root@localhost> \n
DATA. \n
test. \n
```

The experiment consisted of running the automated SMTP client script at the rate of 10 connections per second. We just record the duration for all the first push packet of each connection arriving at the honeypot. The experimental results under different scenarios are shown in Fig 7.

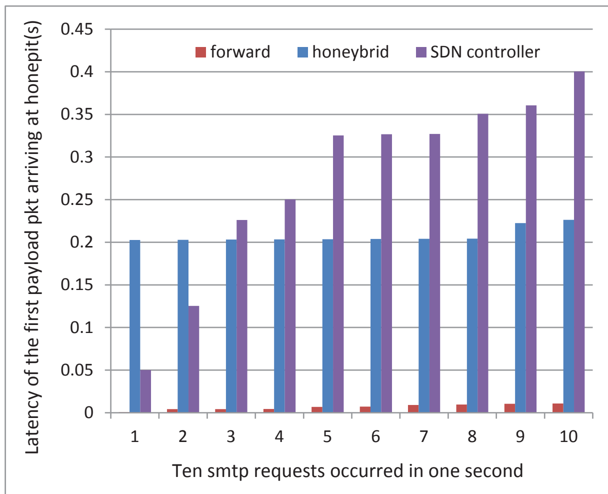

Fig. 7. Connection latency raised by different approaches

The first PSH_ACK packet including payload arriving at the honeypot means the TCP connection between the attacker and honeypot has been established. So the timestamp of the first payload packet arriving at the network interface of the honeypot can be used to calculate the duration for establishing TCP connection. The experimental results show that the connections processed Honeybrid gateway and our SDN controller application can cause much more latency than the normal forward connections. The reason is that both the Honeybrid gateway and the SDN controller need to replay the TCP connection in order to redirection the traffic. However, the delay introduced by the replaying phase of the redirection mechanism is low in our case, as the dialog is made between local systems. Therefore, it will not change importantly the external connection behavior. Furthermore, our SDN controller needs Snort to make decision that results in more latency than the Honeybrid gateway that just uses the rules of IPTABLES to make decision.

Fig. 8 shows the packet I/O graph of honeypot using different mechanisms. The selected interval is 100ms, so the diagram refers to the number of packets is processed by the honeypot in each 100ms. Within this interval, we can

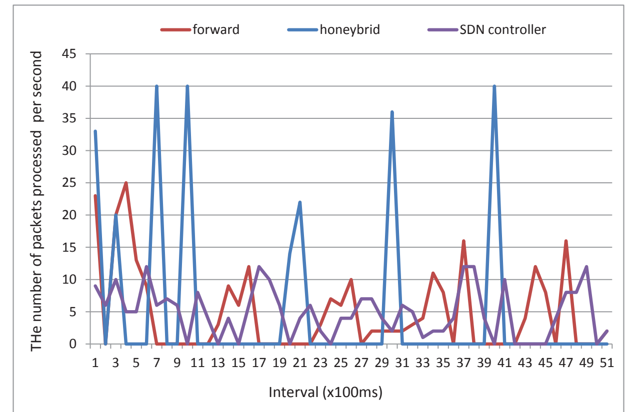

Fig. 8. Packet I/O graph of honeypot under different mechanisms

observe the packet I/O is much more equal-distributed when we use our SDN controller application or never apply any redirection approach. The packet I/O distribution of Honeybrid is much sharper than the other two. Therefore, if the adversary uses the difference of the packet I/O performance to detect the redirection, our mechanism is much stealthier than the Honeybrid gateway's approach.

VI. Conclusion

Honeypots have become a very important security tool for malicious data capture and investigation. However, due to the wide variety of the attacks that take place in real network environments, securing computer systems and investigating the attacks often require the use of sophisticated honeypot systems. The use of individual honeypots is useful when the focus is on some specific attacks, but they have limited application to investigate other types of attacks. The creation and development of hybrid scalable honeypots is aimed to address this problem. A hybrid honeypot system should be able to deploy different types of independent honeypots and control the access of the interesting traffic to the corresponding honeypots according to certain security requirements.

The SDN based network data controller proposed in this paper is aimed to perform the traffic control in hybrid honeypot systems. The controller provides a transparent traffic redirection mechanism that allows forwarding the interesting traffic into corresponding honeypots for further investigation. It implements a TCP connection handover mechanism and a traffic filtering approach based on the use of OpenFlow switches, a Ryu SDN application and the Snort alert mechanism. The use of SDN technologies in this context, specially the facility they provide to programmatically detect and control the data flows in the network has greatly simplified the development of the data controller, compared to other traditional networking approaches.

The preliminary experimental results have allowed to validate the proposal, showing that it can be used to redirect the interesting traffic in a hybrid honeypot systems in a stealthy way. However, some efficiency problems have been detected, mainly due to the limited and simple virtual scenario used for the tests. In the future, we plan to address this inefficiencies and improve the testing scenario, as well as applying the proposed SDN controller to real honeypot systems.

ACKNOWLEDGMENT

This research has been partially supported by the Spanish Ministry of Economy and Competitiveness in the context of GREDOS project (contract no. TEC2015-67834-R) and Elastic Networks (grant no. TEC2015- 71932- REDT).

REFERENCES

[1] M. Nawrocki, M. Wählisch, C. Schmidt, T. C. andKeil, and J. Schönfelder, "A survey on honeypot software and data analysis," *ArXiv e-prints*, Aug. 2016.

[2] N. Provos, "A virtual honeypot framework," in *Proceedings of the 13th Conference on USENIX Security Symposium (SSYM'04)*, Berkeley, CA, USA, 2004, pp. 1–14.

[3] "Dionaea - catched bugs," Nov. 2011. [Online]. Available: http://dionaea.carnivore.it/

[4] X. Jiang and D. Xu, "Collapsar: A vm-based architecture for network attack detention center." in *USENIX Security Symposium*, 2004, pp. 15–28.

[5] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos, "A hybrid honeypot architecture for scalable network monitoring," *Technical Report CSE-TR-499-04, University of Michigan*, 2004.

[6] G. Portokalidis and H. Bos, "Sweetbait: Zero-hour worm detection and containment using low-and high-interaction honeypots," *Computer Networks*, vol. 51, no. 5, pp. 1256–1274, 2007.

[7] H. Artail, H. Safa, M. Sraj, I. Kuwatly, and Z. Al-Masri, "A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks," *Comput. Secur.*, vol. 25, no. 4, pp. 274–288, Jun. 2006.

[8] L. Spitzner, "The honeynet project: trapping the hackers," *IEEE Security Privacy*, vol. 1, no. 2, pp. 15–23, Mar 2003.

[9] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage, "Scalability, fidelity and containment in the potemkin virtual honeyfarm," *ACM Symposium on Operating System Principles (SOSP)*, vol. 39, no. 5, pp. 148–162, Oct 2005.

[10] R. Berthier and M. Cukier, "Honeybrid: A hybrid honeypot architecture," 2008.

[11] H. Welte and P. N. Ayuso, "The libnetfilter_queue project," 2014. [Online]. Available: http://www.netfilter.org/projects/libnetfilter_queue/

[12] T. K. Lengyel, J. Neumann, S. Maresca, B. D. Payne, and A. Kiayias, "Virtual machine introspection in a hybrid honeypot architecture," in *Presented as part of the 5th Workshop on Cyber Security Experimentation and Test*. Berkeley, CA: USENIX, 2012.

[13] W. Fan, D. Fernndez, and Z. Du, "Adaptive and flexible virtual honeynet," in *International Conference on Mobile, Secure, and Programmable Networking*, vol. 9395, Paris, France, June 2015, pp. 1–17.

[14] W. Fan, Z. Du, D. Fernández, and X. Hui, "Dynamic hybrid honeypot system based transparent traffic redirection mechanism," in *17th International Conference on Information and Communications Security (ICICS2015)*, Beijing, China, Dec.9-11 2015, pp. 311–319.

[15] A. Binder, T. Boros, and I. Kotuliak, *A SDN Based Method of TCP Connection Handover*. Cham: Springer International Publishing, 2015, pp. 13–19.

[16] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "Sdnips: Enabling software-defined networking based intrusion prevention system in clouds," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, Nov 2014, pp. 308–311.

[17] P. K. Shanmugam, N. D. Subramanyam, J. Breen, C. Roach, and J. Van der Merwe, "Deidtect: Towards distributed elastic intrusion detection," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing*, ser. DCC '14. New York, NY, USA: ACM, 2014, pp. 17–24.

[18] C. J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 4, pp. 198–211, July 2013.

[19] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238.

[20] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999.

[21] W. Fan, D. Fernndez, and Z. Du, "Versatile virtual honeynet management framework," *IET Information Security*, vol. 11, no. 1, pp. 38–45, 2016.

[22] D. Fernández, F. J. Ruiz, L. Bellido, E. Pastor, W. Omar, and V. Mateos, "Enhancing learning experience in computer networking through a virtualization-based laboratory model," *International Journal of Engineering Education*, vol. 32, no. 6, pp. 2569–2584, December 2016.

[23] W. Fan, D. Fernndez, and V. A. Villagr, "Technology independent honeynet description language," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, Feb 2015, pp. 303–311.